

Genetic and Evolutionary Computation

Stephan Winkler

Leonardo Trujillo

Charles Ofria

Ting Hu *Editors*


Genetic Programming Theory and Practice XX

 Springer

Genetic and Evolutionary Computation

Series Editors

Wolfgang Banzhaf , Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, USA

Kalyanmoy Deb , Department of Electrical and Computer Engineering, Michigan State University, East Lansing, MI, USA

The area of Genetic and Evolutionary Computation has seen an explosion of interest in recent years. Methods based on the variation-selection loop of Darwinian natural evolution have been successfully applied to a whole range of research areas.

The Genetic and Evolutionary Computation Book Series publishes research monographs, edited collections, and graduate-level texts in one of the most exciting areas of Computer Science. As researchers and practitioners alike turn increasingly to search, optimization, and machine-learning methods based on mimicking natural evolution to solve problems across the spectrum of the human endeavor, this growing field will continue to surprise with novel applications and results. Recent award-winning PhD theses, special topics books, workshops and conference proceedings in the areas of EC and Artificial Life Studies are of interest.

Areas of coverage include applications, theoretical foundations, technique extensions and implementation issues of all areas of genetic and evolutionary computation. Topics may include, but are not limited to:

Optimization (multi-objective, multi-level) Design, control, classification, and system identification Data mining and data analytics Pattern recognition and deep learning Evolution in machine learning Evolvable systems of all types Automatic programming and genetic improvement

Proposals in related fields such as:

Artificial life, artificial chemistries Adaptive behavior and evolutionary robotics Artificial immune systems Agent-based systems Deep neural networks Quantum computing will be considered for publication in this series as long as GEVO techniques are part of or inspiration for the system being described. Manuscripts describing GEVO applications in all areas of engineering, commerce, the sciences, the arts and the humanities are encouraged.

Prospective Authors or Editors:

If you have an idea for a book, we would welcome the opportunity to review your proposal. Should you wish to discuss any potential project further or receive specific information regarding our book proposal requirements, please contact Wolfgang Banzhaf, Kalyan Deb or Mio Sugino:

Areas: Genetic Programming/other Evolutionary Computation Methods, Machine Learning, Artificial Life

Wolfgang Banzhaf Consulting Editor BEACON Center for Evolution in Action Michigan State University, East Lansing, MI 48824 USA banzhafw@msu.edu

Areas: Genetic Algorithms, Optimization, Meta-Heuristics, Engineering

Kalyanmoy Deb Consulting Editor BEACON Center for Evolution in Action Michigan State University, East Lansing, MI 48824 USA kdeb@msu.edu

Mio Sugino mio.sugino@springer.com


The GEVO book series is the result of a merger the two former book series: Genetic Algorithms and Evolutionary Computation <https://www.springer.com/series/6008> and Genetic Programming <https://www.springer.com/series/6016>.

Stephan Winkler · Leonardo Trujillo ·
Charles Ofria · Ting Hu
Editors


Genetic Programming Theory and Practice XX


 Springer

Editors

Stephan Winkler 
School of Information
Communications and Media
University of Applied Sciences Upper
Austria
Hagenberg, Austria

Charles Ofria
Department of Computer Science
and Engineering
Michigan State University
East Lansing, MI, USA

Leonardo Trujillo 
Engineering Sciences Graduate Program
Tecnológico Nacional de México
IT de Tijuana
Tijuana, Baja California, Mexico

Ting Hu 
School of Computing
Queen's University
Kingston, ON, Canada

ISSN 1932-0167 ISSN 1932-0175 (electronic)
Genetic and Evolutionary Computation
ISBN 978-981-99-8412-1 ISBN 978-981-99-8413-8 (eBook)
<https://doi.org/10.1007/978-981-99-8413-8>

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Singapore Pte Ltd. 2024

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd. The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Paper in this product is recyclable.

Preface

From 2020 until 2022, the COVID-19 pandemic struck the world and massively changed our daily lives. For the academic world, it meant a sudden halt to in-person meetings, workshops, and conferences, among many other consequences. Of course, several editions of the Genetic Programming Theory and Practice (GPTP) workshop were also affected: In 2020, we had to cancel GPTP; in 2021, we held it online, which turned out great, but still we hoped that we would never be forced to repeat that format for GPTP. In 2022, we planned GPTP as an in-person event at the University of Michigan, and after not being completely sure if it would work for a long time, we were very glad when it finally did! We were very cautious (e.g., wearing masks at the event) and, unfortunately, several of our colleagues who would have preferred to attend the workshop were not able to travel, but overall it was a great event.

November 2022 saw the premiere of the “GPTP Sandbox”, a new online format for GPTP in which we focused on two things: On the first day, six Ph.D. students were given the opportunity to present their research and receive feedback from the community; on the second day, we discussed how genetic programming could be promoted and made more visible in the machine learning community and in society in general.

In 2023, finally, we were able to plan GPTP under more or less “normal” circumstances. (Apart from the fact that for the first time in a long period, Wolfgang Banzhaf was not part of the organization team due to his sabbatical year.) For the first time we planned to hold the workshop at the Kellogg Hotel & Conference Center at Michigan State University, and we also decided to try to integrate more groups that are active in GP research into the GPTP community. Additionally, we also decided to offer the speakers not only the possibility to give regular talks but also so-called lightning talks, i.e., shorter talks in which ideas and research directions could be presented without necessarily having final results or conclusions yet.

And then, in June 2023, GPTP finally came back to Michigan State University! Following the tradition, we started the twentieth edition of our workshop with a joint dinner at a local bar (downtown East Lansing), and then 45 GPTP'ers had three great days at the Kellogg Center, where we had a very nice conference room and delicious food (coffee breaks, lunch, and even a great conference dinner on the first day of the workshop).

Day one started with a great keynote given by Oana Carja from Carnegie Mellon University, she talked about topological puzzles in biology and how geometry shapes evolution and applications to designing intelligent collectives. What a great start to GPTP XX! We then saw and discussed presentations given by Jason Moore and Pedro Ribeiro (Cedars-Sinai Medical Center), Una May O'Reilly (MIT CSAIL), Nathan Haut (Michigan State University), Wolfgang Banzhaf (Michigan State University), and Moshe Sipper (Ben-Gurion University); lightning talks were delivered by Nic McPhee (University of Minnesota), Jose Manuel Velasco Cabo (Universidad Complutense de Madrid), and Matthew Andres Moreno (Michigan State University).

On day two, Thomas Baeck from Leiden University gave an exciting keynote, in which he talked about automated algorithm configuration for expensive optimization tasks; it was very interesting to see all the industrial applications in which theoretical advances of GP have led to successful solutions. Throughout the day, presentations were given by Lee Spector (Amherst College and UMass Amherst), Joel Lehman and Herbie Bradley (Carper.ai), Eric Medvet (University of Trieste) Christian Haider (University of Applied Sciences Upper Austria), and Alexander Lalejini (Grand Valley State University); lightning talks were given by Michael Affenzeller (University of Applied Sciences Upper Austria) and Stuart W. Card.

The last day started with one of the most remarkable keynotes in GPTP history, namely James Foster's talk about his life and his academic journey in computer science and genetic programming. For sure, all of us hearing this talk will never forget it as it was full of interesting stories and emotional moments. We then heard the last talks of GPTP XX, namely lightning talks by Lisa Soros (Barnard College) and Fabricio Olivetti de Franca (Federal University of ABC) and regular talks by Emily Dolson (Michigan State University) and Talib Hussain (John Abbott College).

Throughout the event, not only after the talks but also during breaks and in the evening, we had great discussions, in which we often talked about the current hype machine learning is seeing at the moment—and that GP is the perfect method for so many applications, as it is a very flexible method that produces interpretable results, which is so important in numerous applications. Nevertheless, we all agreed that we have to do more in order to make GP more visible in our community as well as in society in general! One of the most prominent issues we should address is



Fig. 1 Attendees of GPTP XX at Michigan State University, June 2023

that we need more generally and easily available implementations of GP that can be integrated into any data science workflow.

We are very honored and grateful that we could once again organize another GPTP workshop in-person (Fig. 1). It is our intention that GPTP continues to be a core event for genetic programming research, bringing together academics, practitioners and theorists from diverse fields of science that intersect in our community, providing for a constructive, thoughtful, inspired and open interchange of ideas, and to do so, whenever possible, in-person, with a coffee during breaks or a beer at dinner.

Kingston, Ontario, Canada
East Lansing, Michigan, USA
Tijuana, Baja California, Mexico
Hagenberg, Austria
September 2023

Ting Hu
Charles Ofria
Leonardo Trujillo
Stephan Winkler

Acknowledgments

We would like to thank all of the participants for making GP Theory and Practice a successful in-person workshop once again in 2023. Special thanks to our three wonderful keynote speakers, Carja, Thomas, and James: Your talks were amazing!

We would also like to thank our financial supporters for making the existence of GP Theory and Practice possible for twenty great editions. For 2023, we are grateful to the following sponsors:

- Michael Affenzeller from HEAL and the University of Applied Sciences Upper Austria
- Stuart Card
- John Koza
- Jason H. Moore at the Department of Computational Biomedicine in Cedars-Sinai

A number of people made key contributions to the organization of the workshop. We are particularly grateful for contractual assistance by Mio Sugino, Springer-Nature Tokyo, and editorial assistance by Kokila Durairaj, Springer-Nature Chennai. We would also like to express our gratitude to Erik Goodman at the BEACON Center for the Study of Evolution in Action at Michigan State University for his continued support.

Kingston, Ontario, Canada
East Lansing, Michigan, USA
Tijuana, Baja California, Mexico
Hagenberg, Austria
September 2023

Ting Hu
Charles Ofria
Leonardo Trujillo
Stephan Winkler

Contents

1	TPOT2: A New Graph-Based Implementation of the Tree-Based Pipeline Optimization Tool for Automated Machine Learning	1
	Pedro Ribeiro, Anil Saini, Jay Moran, Nicholas Matsumoto, Hyunjun Choi, Miguel Hernandez, and Jason H. Moore	
2	Analysis of a Pairwise Dominance Coevolutionary Algorithm with Spatial Topology	19
	Mario Hevia Fajardo, Per Kristian Lehre, Jamal Toutouh, Erik Hemberg, and Una-May O'Reilly	
3	Accelerating Image Analysis Research with Active Learning Techniques in Genetic Programming	45
	Nathan Haut, Wolfgang Banzhaf, Bill Punch, and Dirk Colbry	
4	How the Combinatorics of Neutral Spaces Leads Genetic Programming to Discover Simple Solutions	65
	Wolfgang Banzhaf, Ting Hu, and Gabriela Ochoa	
5	The Impact of Step Limits on Generalization and Stability in Software Synthesis	87
	Nicholas Freitag McPhee and Richard Lussier	
6	Genetic Programming Techniques for Glucose Prediction in People with Diabetes	105
	J. Ignacio Hidalgo, Jose Manuel Velasco, Daniel Parra, and Oscar Garnica	
7	Methods for Rich Phylogenetic Inference Over Distributed Sexual Populations	125
	Matthew Andres Moreno	

8 A Melting Pot of Evolution and Learning 143
 Moshe Sipper, Achiya Elyasaf, Tomer Halperin, Zvika Haramaty,
 Raz Lapid, Eyal Segal, Itai Tzruia, and Snir Vitrack Tamam

9 Particularity 159
 Lee Spector, Li Ding, and Ryan Boldi

**10 The OpenELM Library: Leveraging Progress in Language
 Models for Novel Evolutionary Algorithms** 177
 Herbie Bradley, Honglu Fan, Theodoros Galanos, Ryan Zhou,
 Daniel Scott, and Joel Lehman

**11 GP for Continuous Control: Teacher or Learner? The Case
 of Simulated Modular Soft Robots** 203
 Eric Medvet and Giorgia Nadizar

**12 Shape-constrained Symbolic Regression: Real-World
 Applications in Magnetization, Extrusion and Data Validation** 225
 Christian Haider, Fabricio Olivetti de Franca, Bogdan Burlacu,
 Florian Bachinger, Gabriel Kronberger, and Michael Affenzeller

**13 Phylogeny-Informed Fitness Estimation for Test-Based Parent
 Selection** 241
 Alexander Lalejini, Matthew Andres Moreno,
 Jose Guadalupe Hernandez, and Emily Dolson

**14 Origami: (un)folding the Abstraction of Recursion Schemes
 for Program Synthesis** 263
 Matheus Campos Fernandes, Fabricio Olivetti de Franca,
 and Emilio Franceschini

**15 Reachability Analysis for Lexicase Selection via Community
 Assembly Graphs** 283
 Emily Dolson and Alexander Lalejini

16 Let’s Evolve Intelligence, Not Solutions 303
 Talib S. Hussain

Index 335

Contributors

Michael Affenzeller Heuristic and Evolutionary Algorithms Laboratory (HEAL), University of Applied Sciences Upper Austria, Hagenberg, Austria

Florian Bachinger Heuristic and Evolutionary Algorithms Laboratory (HEAL), University of Applied Sciences Upper Austria, Hagenberg, Austria

Wolfgang Banzhaf Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, USA

Ryan Boldi University of Massachusetts, Amherst, Amherst, MA, USA

Herbie Bradley CarperAI and EleutherAI and CAML Lab, University of Cambridge and Stability AI, Cambridge, UK

Bogdan Burlacu Heuristic and Evolutionary Algorithms Laboratory (HEAL), University of Applied Sciences Upper Austria, Hagenberg, Austria

Hyunjun Choi Department of Computational Biomedicine, Cedars Sinai Medical Center, Los Angeles, CA, USA

Dirk Colbry Department of Computational Mathematics, Science and Engineering, Michigan State University, East Lansing, MI, USA

Li Ding University of Massachusetts, Amherst, Amherst, MA, USA

Emily Dolson Michigan State University, East Lansing, MI, USA

Achiya Elyasaf Department of Software and Information Systems Engineering, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Mario Hevia Fajardo University of Birmingham, Birmingham, England

Honglu Fan EleutherAI and University of Geneva, Geneva, Switzerland

Matheus Campos Fernandes Federal University of ABC, Santo Andre, SP, Brazil

Fabricio Olivetti de Franca Federal University of ABC, Santo Andre, SP, Brazil

- Emilio Franceschini** Federal University of ABC, Santo Andre, SP, Brazil
- Nicholas Freitag McPhee** University of Minnesota Morris, Morris, MN, USA
- Theodoros Galanos** EleutherAI and University of Malta, Aurecon, Malta, USA
- Oscar Garnica** Universidad Complutense de Madrid, Madrid, Spain
- Christian Haider** Heuristic and Evolutionary Algorithms Laboratory (HEAL), University of Applied Sciences Upper Austria, Hagenberg, Austria
- Tomer Halperin** Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel
- Zvika Haramaty** Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel
- Nathan Haut** Department of Computational Mathematics, Science and Engineering, Michigan State University, East Lansing, MI, USA
- Erik Hemberg** ALFA, MIT CSAIL, Cambridge, England
- Jose Guadalupe Hernandez** Michigan State University, East Lansing, MI, USA
- Miguel Hernandez** Department of Computational Biomedicine, Cedars Sinai Medical Center, Los Angeles, CA, USA
- J. Ignacio Hidalgo** Instituto de Tecnología del Conocimiento, Universidad Complutense de Madrid, Madrid, Spain
- Ting Hu** School of Computing, Queen's University, Kingston, ON, Canada
- Talib S. Hussain** John Abbott College, Ste-Anne-de-Bellevue, QC, Canada
- Gabriel Kronberger** Heuristic and Evolutionary Algorithms Laboratory (HEAL), University of Applied Sciences Upper Austria, Hagenberg, Austria
- Alexander Lalejini** Grand Valley State University, Allendale, MI, USA
- Raz Lapid** DeepKeep, Tel-Aviv, Israel
- Joel Lehman** CarperAI and StabilityAI, Newark, NJ, USA
- Per Kristian Lehre** University of Birmingham, Birmingham, England
- Richard Lussier** University of Minnesota Morris, Morris, MN, USA
- Nicholas Matsumoto** Department of Computational Biomedicine, Cedars Sinai Medical Center, Los Angeles, CA, USA
- Eric Medvet** Department of Engineering and Architecture, University of Trieste, Trieste, Italy
- Jason H. Moore** Department of Computational Biomedicine, Cedars Sinai Medical Center, Los Angeles, CA, USA

Jay Moran Department of Computational Biomedicine, Cedars Sinai Medical Center, Los Angeles, CA, USA

Matthew Andres Moreno University of Michigan, Ann Arbor, MI, USA

Giorgia Nadizar Department of Mathematics and Geosciences, University of Trieste, Trieste, Italy

Gabriela Ochoa Department of Computer Science, University of Stirling, Stirling, UK

Una-May O'Reilly ALFA, MIT CSAIL, Cambridge, England

Daniel Parra Universidad Complutense de Madrid, Madrid, Spain

Bill Punch Department of Computational Mathematics, Science and Engineering, Michigan State University, East Lansing, MI, USA

Pedro Ribeiro Department of Computational Biomedicine, Cedars Sinai Medical Center, Los Angeles, CA, USA

Anil Saini Department of Computational Biomedicine, Cedars Sinai Medical Center, Los Angeles, CA, USA

Daniel Scott EleutherAI and Georgia Institute of Technology, Atlanta, USA

Eyal Segal Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Moshe Sipper Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Lee Spector Amherst College, Amherst, MA, USA;
University of Massachusetts, Amherst, MA, USA

Snir Vitrack Tamam Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Jamal Toutouh University of Malaga, Malaga, Spain

Itai Tzruia Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Jose Manuel Velasco Universidad Complutense de Madrid, Madrid, Spain

Ryan Zhou EleutherAI and Queen's University, Kingston, Canada

Chapter 1

TPOT2: A New Graph-Based Implementation of the Tree-Based Pipeline Optimization Tool for Automated Machine Learning



Pedro Ribeiro, Anil Saini, Jay Moran, Nicholas Matsumoto, Hyunjun Choi,
Miguel Hernandez, and Jason H. Moore

1.1 Introduction

In recent years, machine learning (ML) has been applied to a number of domains, including image recognition, weather forecasting, stock market prediction, recommendation engines, text generation, etc. A whole gamut of algorithms is available for these tasks, such as Logistic Regression, Naive Bayes, K-Nearest Neighbors, Decision Tree, Random Forest, Support Vector Machine, etc. Each algorithm also has a large number of hyperparameter settings to adjust. In addition, a typical user also needs to select methods for data cleaning, feature selection, feature engineering, etc. The role of a data scientist is to search through a large space of possible operators and their hyperparameters in order to find the best-performing pipeline for a given task.

Over the years, multiple methods have been developed to automate searching for the best machine learning pipeline. One of those methods is TPOT [13]. In TPOT, the pipelines are represented as trees, where the root node is either a classifier or a regressor, with other nodes encoding other ML operators for data preprocessing, feature engineering, etc.

Over the years, TPOT has been successfully applied to several problems, such as genetic analysis [12]. Several extensions of TPOT have been developed that optimize the existing implementation or provide additional functionality. However, these changes were often not merged into the main software package. For example, in Parmentier et al. [14], the authors fork TPOT and implement a successive halving strategy to reduce computational demand. The algorithm begins by quickly searching through a larger population evaluated on smaller portions of the data early in the training phase, then evaluating fewer but likely better-performing models on larger portions of the dataset later. This allowed TPOT to explore a larger search space as

P. Ribeiro (✉) · A. Saini · J. Moran · N. Matsumoto · H. Choi · M. Hernandez · J. H. Moore
Department of Computational Biomedicine, Cedars Sinai Medical Center, Los Angeles, CA, USA
e-mail: pedro.ribeiro@cshs.org

© The Author(s), under exclusive license to Springer Nature Singapore Pte Ltd. 2024
S. Winkler et al. (eds.), *Genetic Programming Theory and Practice XX*,
Genetic and Evolutionary Computation,
https://doi.org/10.1007/978-981-99-8413-8_1

well as reach better performance. TPOT was forked and modified in another paper to use covariate adjustments [11]. Recently, TPOT has been forked and modified to automate quantitative trait locus (QTL) analysis in a biology-based AutoML software package called AutoQTL [9]. However, none of the above forks of TPOT have been merged into the master branch. Due to the way TPOT and these forks are structured, it would also be difficult to do so.

Since these extensions have been developed in isolation, they may not be compatible with the main repository. This limits their usability as different features might not be used together, and users may not be aware of the different forks and their respective features.

In this chapter, we introduce a new edition of TPOT called TPOT2.¹ Although TPOT2 shares some similarities with TPOT in how it searches for ML pipelines, it has been implemented from scratch to be easily maintainable and extendable. TPOT2 uses a graph-based representation which is much more flexible than the tree-based representation used by TPOT. TPOT2 also gives the user much more flexibility to define various parameters of the underlying algorithm. The following sections describe the TPOT2 algorithm, including the representation, genetic operators, etc. For the remainder of the paper, we will refer to the original TPOT as TPOT1 and the new version as TPOT2.

1.2 Related Work

There are several other methods in the domain of AutoML, which differ in the way they search for optimal machine learning pipelines. Some popular methods include Auto-WEKA [16], Auto-Sklearn 1 and 2 [5, 7], and TPOT [13], among others. Both Auto-Weka and Auto-Sklearn, for example, use Bayesian optimization. Auto-Sklearn also improves upon the performance of its earlier version with meta-learning, successive halving, and other optimizations. TPOT utilizes an evolutionary algorithm to search through the space of possible pipelines.

Other existing packages have inspired various parts of TPOT2. For example, several Python packages implement an API for evolutionary algorithms, not necessarily to evolve machine learning pipelines. Some popular packages include PyMoo [2], DEAP [8], and KarooGP [3], etc. Other packages, such as `baikal`² and `skdag`,³ have been developed for easily building graph-based Scikit-Learn pipelines.

Instead of using the existing implementations for the evolutionary algorithm and graph-based pipelines, we developed a new implementation to meet our needs. In the future, we may work toward adding the functionality of exporting to other graph pipeline representations into TPOT2.

¹ <https://github.com/EpistasisLab/tpot2>.

² <https://github.com/alegonz/baikal>.

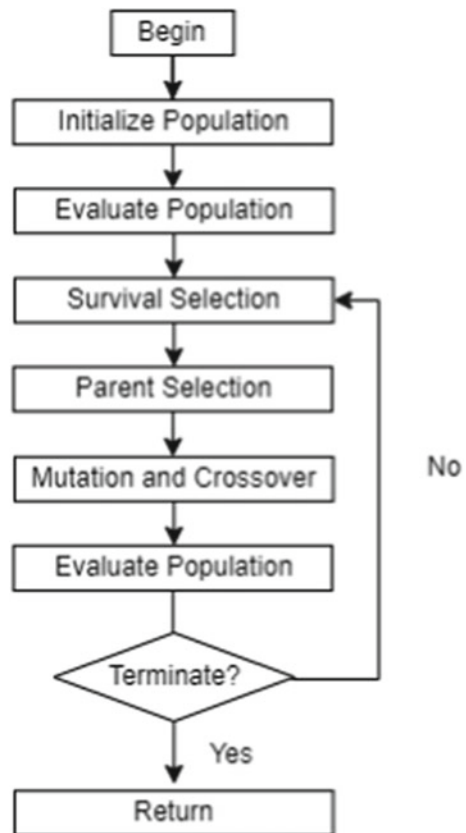
³ <https://github.com/scikit-learn-contrib/skdag>.

1.3 Evolutionary Algorithm

TPOT2 implements an evolutionary algorithm module that follows a standard algorithm outlined in Fig. 1.1. First, an initial population is generated and evaluated. The individuals in the initial population are generated sequentially in the following way. TPOT2 loops through the possible final estimators and assigns them as the root node to the individuals one by one. After all final estimators have been assigned as a root to different individuals, the loop repeats for the following individuals. We randomly add between 0 and $\max(i, 3)$ nodes to each root in the i th loop. We stop the loop when the required number of individuals has been generated.

The population is a list of individuals that is available to be used to generate new individuals. The primary evolutionary algorithm loop begins with survival selection, where lower-performing individuals are removed from the current population. Next, the parent selection algorithm selects (with replacement) individuals to be used in mutation, crossover, or a combination of both. Next, the crossover and mutation methods are used to generate new individuals. Note that crossover is one-directional,

Fig. 1.1 A flowchart illustrating the evolutionary algorithm used in TPOT2



which means it generates one individual per pair. More details on mutation and crossover operators are found in the next section. If a newly generated individual is identical to an already evaluated individual, TPOT2 will mutate that individual until it becomes unique (up to 20 attempts).

With parallel processing, we want to ensure each node or core has an individual to evaluate every generation. This allows us to ensure we are creating the expected number of individuals to saturate the computational resources. Finally, the set of new individuals is evaluated. Individuals that throw errors or time out are discarded. The remainder gets added to the current population list. The algorithm then loops back into survival selection, where the now expanded population is cut down, and the loop continues. We exit the loop once we complete the desired number of generations, satisfy an early stopping condition, or receive a manual termination signal. The default best individual is the one with the highest value of the first objective function, which by default, is the cross-validation score on the training data.

TPOT1 and TPOT2 use the Nondominated Sorting Genetic Algorithm (NSGA-II) for survival selection [4]. For parent selection, TPOT selected parents randomly from the population, whereas the default in TPOT2 is to use Dominated Tournament Selection (as described in the NSGA-II paper [4]). While the selection methods are hardcoded into TPOT1, they can be passed as parameters in TPOT2.

1.4 GraphPipelineIndividual Representation

The individuals in TPOT2 are represented as NetworkX directed acyclic graphs [10]. Figure 1.2 shows an example TPOT2 individual. The GraphPipelineIndividual class contains a template for the pipeline represented as a directed acyclic graph. A single node in the graph contains both the machine learning method type and its hyperparameters. The individual holds other parameters that dictate their search space:

- *root_config_dict*: Defines the root node's possible methods and hyperparameter ranges.
- *inner_config_dict*: Defines the inner nodes' possible methods and hyperparameter ranges. If set to *None*, the graph will have no inner nodes.
- *leaf_config_dict*: Defines the leaf nodes' possible methods and hyperparameter ranges. If set to *None*, leaf nodes are pulled from *inner_config_dict*.
- *max_size*: The maximum number of nodes in any pipeline.
- *linear_pipeline* : If *True*, TPOT2 will evolve only linear pipelines.

The structure of configuration dictionaries (*root_config_dict*, etc.) is different from the one in TPOT1. The keys are the Python Types for the desired method, and the corresponding values are functions that return a set of hyperparameters from the desired search space. These functions are designed to be compatible with the Optuna hyperparameter optimization package [1]. The hyperparameters are currently chosen randomly, but we plan to explore Optuna integration in the future.

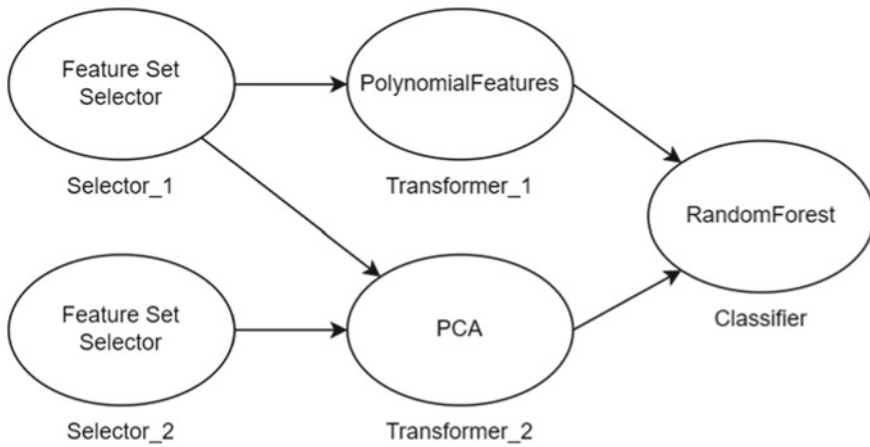


Fig. 1.2 An example individual in TPOT2

Additionally, when setting the key to be the special-case string “Recursive,” the user can pass in the above parameters in a dictionary in place of a function to recursively define the search space for a given node. For example, the user can define the configuration dictionaries in such a way that the leaf nodes are set to be a pipeline of the shape “Selector-Transformer-Classifer” and the root node is a final classifier.

These configuration dictionaries can be customized for specific tasks. Some examples of useful search spaces users can define using the above tools are:

- **AutoQTL:** AutoQTL [9] is a fork of TPOT with a custom configuration and objective function. This can be replicated in TPOT2 by using a custom configuration dictionary that includes the genetic encoders and encoding frequency selectors used as transformers and selectors, as well as passing in the custom objective function as described in the paper.
- **Logistic regression with Selected Features:** TPOT2 can perform genetic feature selection for a given machine learning algorithm. For example, the root node can be set to logistic regression; all leaf nodes can be set to a set of feature selectors. Then TPOT2 can evolve a set of selectors that optimize the performance of logistic regression.
- **Symbolic regression or classification:** TPOT2 can also evolve symbolic regression or classification pipelines. The root node can be set to linear or logistic regression, inner nodes can encode basic arithmetic operators, and leaf nodes can encode Feature Set Selectors.

1.4.1 Mutation

In TPOT2, we implement eight mutation methods. Currently, each of these methods is selected with uniform probability during mutation. In the future, we might explore applying these methods with different probabilities.

- **`_mutate_get_new_hyperparameter`**: Pick a node randomly and assign new hyperparameters to that node.
- **`_mutate_replace_method`**: Pick a node randomly and select a new operator (and hyperparameters) for that node.
- **`_mutate_remove_node`**: Pick a random node (other than the root node) and remove it from the graph. Connect all its children to all its parents.
- **`_mutate_remove_extra_edge`**: Randomly pick a node with more than one outgoing edge and randomly remove one edge.
- **`_mutate_add_connection_from`**: Pick two nodes and add an edge between them (as long as the graph could still be acyclic after the addition).
- **`_mutate_insert_leaf`**: Create a new node and add an edge between it and the randomly chosen existing node.
- **`_mutate_insert_bypass_node`**: Pick two nodes at random and create a new node. Add edges from one of the existing nodes to the new node and from the new node to the second node (as long as the graph could still be acyclic after the addition).
- **`_mutate_insert_inner_node`**: Pick two nodes connected by an edge and create a new node. Remove the edge between the original nodes. Add edges from one existing node to the new node and from the new node to the second existing node.

1.4.2 Crossover

The crossover operator in TPOT2 takes in two individuals and modifies the first individual.

- **`_crossover_swap_branch`**: A branch, or a subgraph, is a node and its descendants. This operator selects the root of a subgraph and removes it. All other nodes in the subgraph are removed if they become disconnected from the root of the whole graph. (If a node in the graph has another path to the root, it is not removed). A full subgraph from the second individual is copied into the first individual with outgoing edges to the same parents as the originally selected node. This is similar to a subtree crossover generalized to directed acyclic graphs.
- **`_crossover_take_branch`**: This operator copies a branch (subgraph) from the second individual and attaches an edge from the root of the branch to a node chosen at random in the first individual.

1.5 TPOT2 API

1.5.1 *TPOTEstimator*

The `TPOTEstimator` class is the primary entry point into TPOT2, where users can define the search space and input other parameters for evolving graph pipelines. For convenience and consistency with TPOT1, `TPOTClassifier`, and `TPOTRegressor` classes are also provided, which contain default parameters for use in classification and regression tasks, respectively.

TPOT2 exposes more parameters and options than TPOT1, providing users more flexibility in defining the search space and evolutionary parameters. For example, users can now provide their own objective functions, specify the maximum size of the pipelines, separately define the possible operators for root nodes, inner nodes, and leaf nodes, and even define a set of preprocessing steps for all pipelines. Additionally, there are parameters related to the evolutionary algorithm, such as survival selection methods, parent selection methods, genetic operators, changing population size over time, changing the proportion of data used over time, etc.

1.5.1.1 Optimizations

In Parmentier et al. [14], the authors describe how their successive halving can improve the performance of TPOT1. It has also shown to be very successful in Auto-Sklearn. Unfortunately, this feature was never brought into the main software package in TPOT1. We re-implement this feature in TPOT2. The premise of the algorithm is to evaluate a large number of pipelines with a small subset of the data in the early generations and fewer pipelines with a larger subset or all of the data in the later generations.

The user can provide the range as well as the rate of change of values of the corresponding parameters. The original paper found that halving and doubling population size and computational budget a few times over the course of the evolution led to performance improvements. Future work can look into other ways of scaling it.

1.5.1.2 Cross Validation Early Stopping (Experimental)

We can reduce the computational load by not fully evaluating all folds in cross-validation in poorly performing models. When a model performs poorly on the first few folds of cross-validation, it can be reasonably assumed that it will continue to perform poorly on the remaining. We can save time and computational resources by terminating the evaluation early.

TPOT2 implements two strategies; both can be used independently or simultaneously.

- **Threshold Early Stopping:** After evaluating each fold, the pipeline must reach a certain percentile of all previously evaluated scores; otherwise, it is discarded.
- **Selection Early Stopping:** We evaluate each fold one at a time in selecting early stopping. After each fold, we select the best individuals and discard the rest.

In the future, we will look into implementing an algorithm similar to greedy k-fold cross-validation as described by Soper [15] to evaluate a population more efficiently.

1.5.1.3 Validation Set to Select From Pareto Front Models

TPOT1 would sometimes overfit the cross-validation score with overly complex pipelines that had poor generalization compared to the simpler pipelines in the Pareto front. This is more common in smaller datasets. TPOT2 can subsample the training data into a validation set. It then uses the validation set to select the best model from the Pareto front, hopefully avoiding overfit models.

1.5.2 Ensembling

TPOT1 includes classifiers and regressors in the search space for the inner nodes. All inner classifier and regressor operators (i.e., those that are not the final estimator) were wrapped in a *StackingEstimator* object which passed through its inputs in addition to its prediction to the next operator. When two classifiers or regressors exist in different branches, however, this would cause two duplicates of the data to be passed to the final operator. This could negatively impact performance.

In TPOT2, there is no passthrough from classifiers or regressors. Only the model predictions pass to the next layer. This allows TPOT2 to learn whether or not to pass through data to the root (final estimator) node.

In testing, we have not found much improvement in including classifiers and regressors in inner nodes for TPOT2. By default, only transformers and selectors are included in the search space for the inner nodes. Future work will explore strategies to improve the ensembling strategies in TPOT2. It is possible that optimizations (in parallelization, caching, successive halving, early termination of the cross-validation evaluation, etc.) may improve the performance of ensembling by allowing more models to be evaluated in the same time period. Another option would be to do post hoc ensembling with the best-evaluated pipelines after the evolutionary algorithm completes. There may be an optimal configuration dictionary to define an efficient search space for an ensemble. For example, an individual could be defined by an ensemble of ‘selector-transformer-classifier’ pipelines followed by a meta-classifier.

1.6 Experiment Set-Up

For our experiments, we use the benchmark datasets from diverse domains compiled in Feuer et al. [6] and hosted my OpenML. Specifically, we compare the performance of TPOT2⁴ against TPOT1 on the 39 OpenML tasks grouped in D_{test} in that paper. Each task provides a training and testing split for an underlying dataset. These datasets vary in the number of samples, the number and types of features, as well as the presence of missing values.

To make the comparison between indexTree-Based Pipeline Optimization Tool (TPOT) TPOT1 and TPOT2 as fair as possible, we do some preprocessing steps. To take care of the missing values, we preprocess the data with mean imputation of the numeric variables and mode imputation of the categorical variables. Then, we perform one-hot encoding of categorical columns with the minimum frequency of categories set to 0.001. The latter was done since TPOT1 does not have a parameter for specifying categorical columns and does not do one-hot encoding by default. The preprocessing resulted in datasets with the number of samples ranging from 463 to 389279 and the number of columns ranging from 4 to 7200.

To ensure the same partitions of the data for cross-validation in TPOT1 and TPOT2, we pass in a scikit-learn cross-validation splitter with randomized splits and the same seed to both.

Note that OpenML task 168795 partitions its data such that there are less than ten examples for two classes in the training set. We, therefore, randomly resampled the classes with fewer than ten examples and appended them to the training data so that for each class we have at least 10 samples. This was done so that it could be correctly split during cross-validation. Additionally, the dataset for task 189866 was very large and causing TPOT1 and TPOT2 to run into memory issues. To alleviate memory issues for this experiment, we set `n_jobs` to 24 rather than 48 and allowed 10 h instead of 5 h before considering the run to have timed out.

All experiments were conducted on a High-Performance Computing cluster node with an Intel Xeon Gold 6342 CPU with 48 threads and 1TB of memory.

1.6.1 TPOT1 Versus TPOT2

Although TPOT1 and TPOT2 have slightly different parameters, we set parameter values for them so that experimental settings for them are as close to each other as possible. The parameter values are shown in Table 1.1. We compare the performance of TPOT1 and TPOT2 at 30 generations. A population size of 48 was selected to match the 48 threads on the CPU we are using. Each individual pipeline is given a limit of 5 min to be evaluated. In theory, this should take approximately 5 min per generation since all individuals can be evaluated simultaneously. The maximum time for the entire process would be approximately 2.5 h, with some extra time

⁴ The code to run experiments: https://github.com/epistasisLab/tpot2_gptp_experiments.

Table 1.1 AutoML methods and their parameters

Method	Parameter	Values
TPOT1	scoring	“roc_auc” for Binary, “neg_log_loss” for multiclass
	population_size	48
	generations	30
	n_jobs	48
	cv	StratifiedKfold(n_splits=10, shuffle=True, random_state=42)
	max_time_mins	None
	max_eval_time_mins	5
TPOT2	scores	[“roc_auc”] for Binary, [“neg_log_loss”] for multiclass
	n_jobs	48
	cv	StratifiedKfold(n_splits=10, shuffle=True, random_state=42)
	max_eval_time_seconds	300
	crossover_probability	0.1
	mutate_probability	0.9
	mutate_then_crossover_probability	0
	crossover_then_mutate_probability	0
	other_objective_functions	[number_of_nodes_objective]
	other_objective_functions_weights	[-1]
	memory_limit	None
	preprocessing	False

allocated for processing between generations and the final pipeline fitting. We also set a maximum time limit of 5 h per run to allow for wiggle room. If the run exceeds this limit, it will be terminated and not included in the results. Both algorithms used a 90% mutation rate and a 10% crossover rate.

The primary objective function we use is: maximize the area under the receiver operator curve (auROC) for binary problems or minimize the log loss for multiclass problems. In addition, a secondary objective function was included that minimizes the number of nodes in a given pipeline.

There is a key difference in the search space of the two algorithms. TPOT1 allows classifiers wrapped inside a custom StackingEstimator class, which passes through its inputs along with its predictions to the next node, to be included in the inner or leave nodes (this cannot be disabled). TPOT2, however, for these experiments, includes classifiers only in the root node.

1.7 Results and Discussion

For each dataset, we evaluated five runs each for TPOT1 and TPOT2. Table 1.2 summarizes the number of completed and failed runs for each method. A failed run is one in which the algorithm throws an error and terminates prematurely, or does not finish in the allotted time period and is killed by the system. TPOT1 failed on 45 runs, and TPOT2 failed only on one. The method that TPOT1 utilizes for timing out pipelines does not always work correctly, partly due to the fact that it is incompatible with the C backends of many algorithms. This would cause it to get stuck and run over time. Some TPOT1 failures were also due to memory constraints with the larger datasets. In some other cases, TPOT1 seems to abruptly end training early before completing all generations. We are not sure what the cause of this is. We still included those results in our analysis as it returned a final pipeline.

The runs where TPOT2 failed were all on the largest dataset due to memory issues. There are still some optimizations and fixes to be made so that it is more stable for larger datasets. TPOT2 may also effectively end training early when memory issues prevent it from continuing training. This occurs when all worker processes crash simultaneously, generally due to memory issues. This then causes all subsequent evaluations to fail (Fig. 1.3).

Table 1.2 Summary of run completion for each algorithm

Algorithm	failed_count	completed_count
TPOT1	45	150
TPOT2	1	194

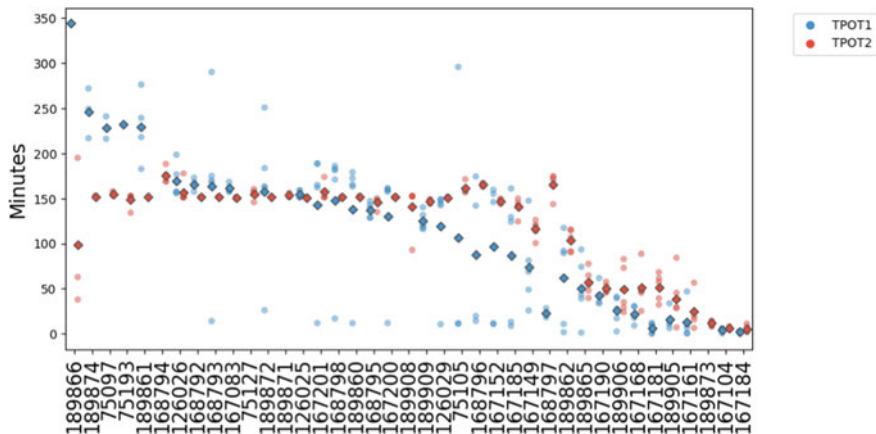


Fig. 1.3 Stripplot summarizing duration in minutes to run each method on a given dataset. The x-axis denotes the identifiers of the datasets

We later found a bug introduced in version 0.12.0 that caused early termination of some runs in TPOT1. This bug reduced training times for some runs and potentially lowered the average scores for TPOT1 runs in this paper. We have fixed this particular bug in the 0.12.1 release. The updated results with the fixed TPOT1 are published in the GitHub repository.

Note that we only include completed runs (including the TPOT1 runs that end training abruptly but still return a final pipeline) in our analyses in this paper.

The performance of TPOT1 and TPOT2 on each dataset is measured in the following way. For every run, we take the pipeline with the best objective function value (*log loss* or *auroc*, based on the dataset) across all generations on the training set, calculate its performance on the holdout set, and report that value. We average the values for five runs. Figure 1.4 illustrates the scores of the best pipeline found in a

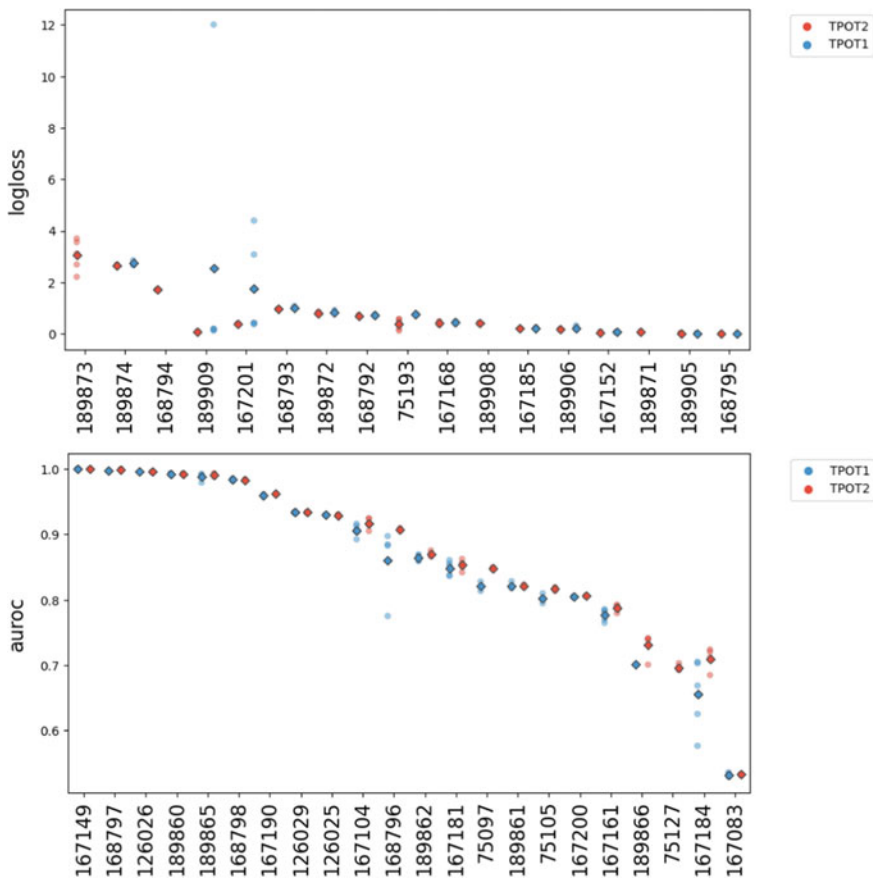
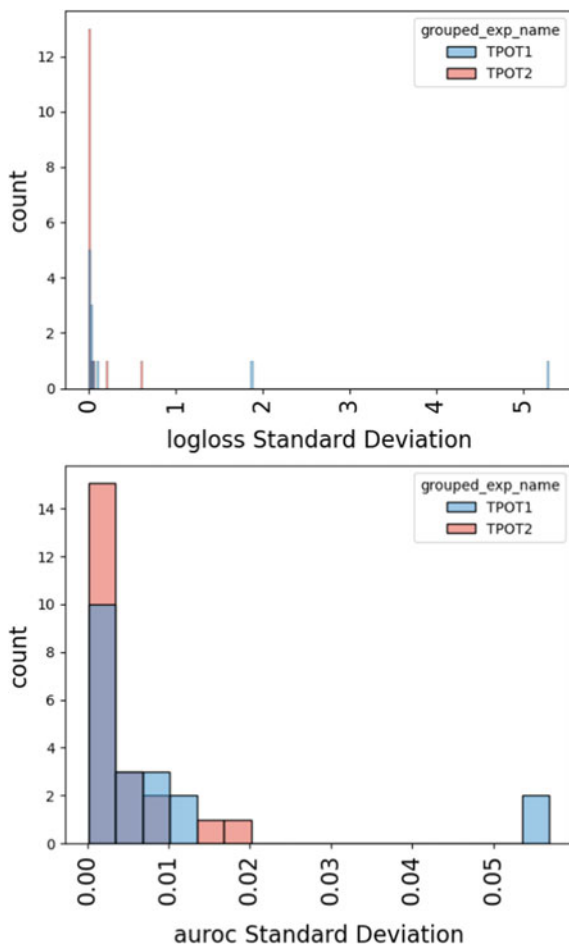


Fig. 1.4 Stripplot summarizing the test scores of both methods on all the datasets. Log loss scores are reported for multiclass tasks and AUROC for binary problems. Diamonds indicate the average score for a method on a particular dataset

Fig. 1.5 Histograms (for binary and multiclass datasets, respectively) summarizing the standard deviation of scores for completed runs across all datasets



given run on the holdout set for each dataset. Additionally, in Fig. 1.5, we show the standard deviation of scores for each model across all datasets.

In Fig. 1.6, we plot the average test scores of TPOT1 and TPOT2 on all the datasets. Since all the points are very close to the diagonal line, we see that TPOT1 and TPOT2 have very similar performances with the given parameter setting though in a few instances, TPOT2 had meaningfully better scores than TPOT1.

Next, we take a deeper look into the results of the TPOT2 runs. Figure 1.7 plots the best cross-validation scores on the training set in the population for a given generation for all runs on different datasets. TPOT2 appears to generally converge quickly, with only minor improvements after a few generations.

We also look at the Pareto front of all the evaluated pipelines in a given run. This is shown in Fig. 1.8, where the y-axis denotes the values of the primary objective (log loss or auROC), and the x-axis denotes the value of the secondary objective (number of

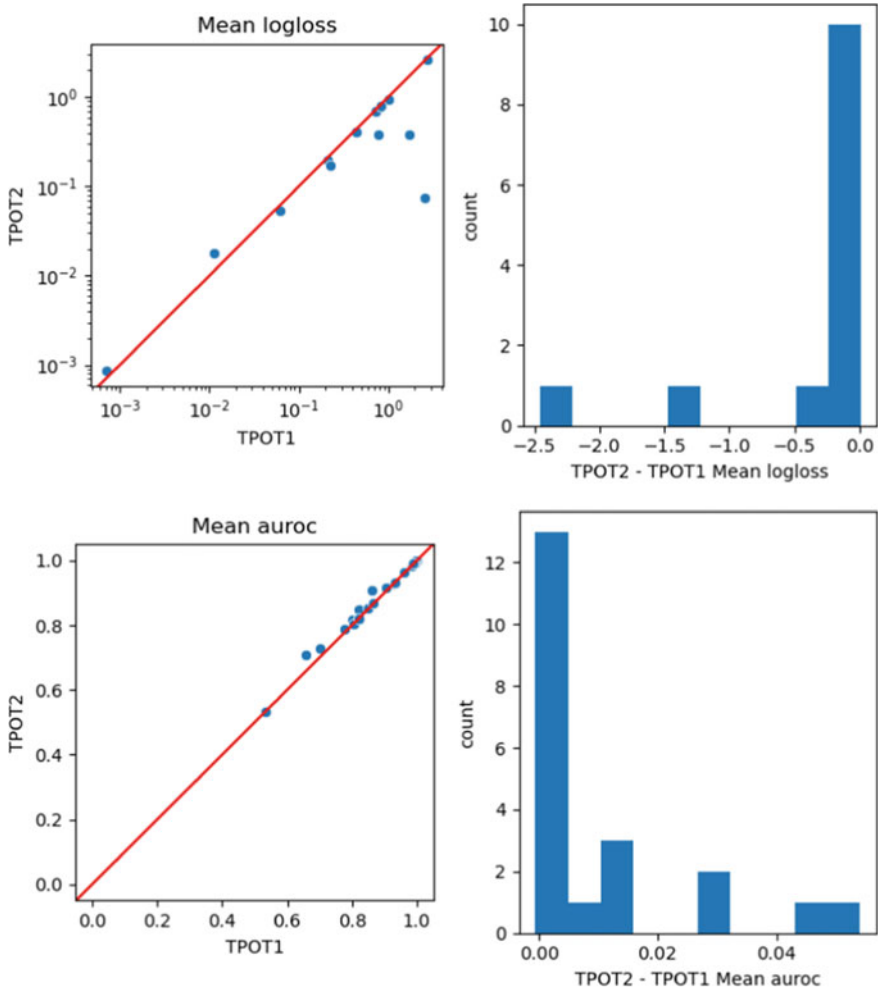


Fig. 1.6 Scatterplot summarizing the test scores of the methods on all the datasets. Each point in the plots represents a dataset. Log loss scores are reported for multiclass tasks and AUROC for binary problems

nodes in a given pipeline). The plots show that the larger models provide negligible improvements in primary objective scores.

The number-of-nodes objective is not a perfect measure of model complexity. A larger pipeline may be simpler than a smaller one. For example, a pipeline with logistic regression and several feature selectors is likely simpler than a model with a single XGBoost. However, more often than not, larger pipelines are usually more complex and overfit the cross-validation score on the training data. TPOT1 often had a similar issue where the final returned model was large. While the large model

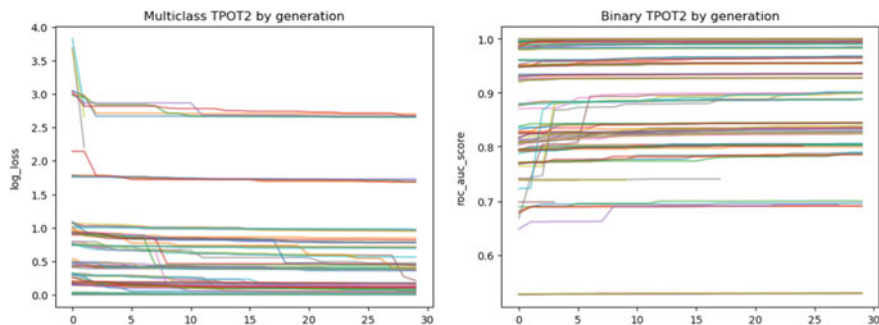


Fig. 1.7 TPOT2 best cross-validation scores in the population for a given generation. Each line represents a single run for a given dataset

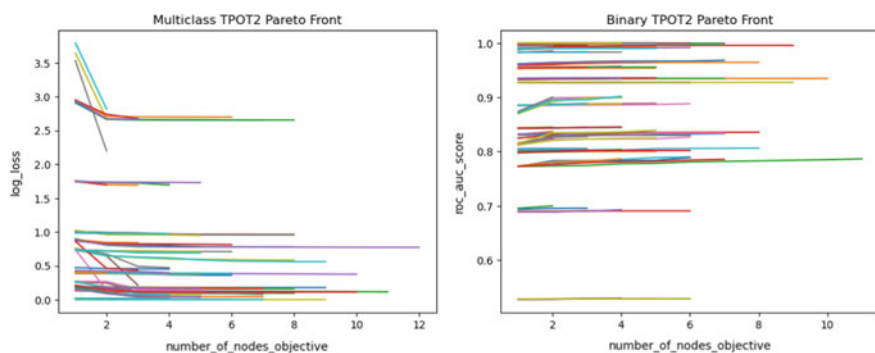


Fig. 1.8 Pareto front of all the evaluated pipelines. Each line represents the pareto front for a single run

would have the best cv score, it would underperform on test data compared to other pareto front models.

1.8 Conclusions

In this chapter, we presented a new version of the popular AutoML package TPOT called TPOT2. Among other differences, TPOT2 uses graph-based representation for individuals instead of the tree-based representation used by TPOT. By benchmarking both versions on a diverse set of datasets from OpenML, we show that the new version performs at least as well as the original. Moreover, the new version has several additional features that allow it to be more flexible for different types of problems, more easily maintained, and more easily extended. We will continue to build additional features and optimizations into TPOT2.

1.8.1 Future Work

There are many avenues for optimizing existing and adding new features in TPOT2. In this section, we highlight some of our future plans.

1.8.1.1 MetaLearner

TPOT2 allows the users to specify the search space of pipelines for a particular dataset and control other aspects of evolutionary search through many different parameter settings. However, different parameter settings will be optimal for different types of problems. For example, a large dataset might benefit more from using the successive halving algorithm than a relatively smaller one. However, trying out several parameter settings can be computationally expensive. Autosklearn2 addresses this issue by training a ‘meta-learner’ that estimates the best parameter values for a given dataset. In the future, we can also look into training a meta-learner to estimate optimal parameters for a given dataset in TPOT2.

1.8.1.2 Optuna Optimization

Currently, hyperparameters for different ML operators in TPOT2 are generated and mutated randomly. We plan to look into different strategies for integrating Optuna to optimize hyperparameters during an evolutionary run.

1.8.1.3 Interpretability

Given that the increase in the performance of TPOT2 plateaued early in an evolutionary run on many datasets, it is possible that complex pipelines are not required for optimal performance on these datasets. In order to further improve the performance, future work could look into having TPOT2 focus more on hyperparameter optimization in the later generations. Alternatively, we could leverage TPOT2’s strength in complex graph building to try to build more interpretable pipelines that may be composed of a higher number of more interpretable steps. For example, optimizing a set of more robust feature selections and engineering followed by a simple classifier as opposed to a single complex XGBoost model. This process could include defining an objective function that more accurately measures interpretability.

Acknowledgements The study was supported by the following NIH grants: R01 LM010098 and U01 AG066833.